Ohio Network Emulator (ONE) Enhancements for Satellite Network Research

James McKim, Robert Dimond

Abstract— **The ONE application is a network emulator that can be run on an off the shelf computer workstation. ONE provides an inexpensive way to perform the kinds of network emulation of interest in space network research. Effects that would be otherwise difficult or impossible to simulate (i.e. communication transit delays as seen by satellites and spacecraft) can be studied in the lab as new protocols are developed. ONE runs entirely in user space and doesn't require any driver or kernel modifications.**

ONE [1] was initially developed at Ohio University and has since undergone major revision at NASA Glenn Research Center Satellite Network and Architectures branch to enhance the capabilities of ONE pertinent to the space based Internet protocol research.

Of most interest is the variable propagation delay capability. The software currently supports propagation delays as defined by output from Satellite Toolkit (STK) simulation runs. Interfaces to other sources of propagation delay data can easily be added as needed. Tests of the effects of changing propagation delays on the performance of networks can be done. This will facilitate development of strategies to efficiently and reliably handle network environments like those encountered by spacecraft or satellites.

Other capabilities include: Random Early Detection (RED), an active queue management strategy that attempts to improve performance and reduce the average queue length, and Bit Error Rate (BER) simulation (the injection of simulated noise into the data stream). The design is amenable to inclusion of other software controlled effects.

The ONE application is being made available to outside researchers in a open source format. The product will be supported internally, and ideally, benefit from having the wider external distribution.

I. Introduction

The Satellite and Networks Architecture Branch at NASA Glenn Research Center has adapted the ONE (Ohio Network Emulator) application to provide a useful platform on which to perform satellite network research.

ONE emulates a link between a pair of network interfaces. Various effects simulating those found in the real world can be applied to traffic flowing between the two interfaces. A report on an earlier version of ONE was written [1]. Capabilities have been enhanced since that report was issued.

Data networks that include satellite and spacecraft nodes impose effects on network traffic not experienced in terrestrial networks. The function of ONE is to emulate networks, including those using satellites in order to provide an inexpensive way to develop and test network protocols.

As an example: in networks that include satellites, propagation delay can be significantly longer than is experienced in terrestrial networks. The increased latency can have unexpected effects on traffic through the network, utilization of the network, and consequently, applications communicating via the network.

One additional effect on traffic associated with many space based networks: at intervals and for periods ranging upwards of several minutes to many hours, differing orbits cause line of sight is obscured and subsequent temporary loss of a link.

II. Design

ONE runs in user space. It requires no changes to the kernel or device drivers.

At the core of the ONE application is a loop within which packets are moved from input to output queues. Incoming traffic is monitored on each interface and routed the the other interface's output queue. Depending on the configured propagation delay packets are timestamped as to when they should be transmitted. Checks are also performed as to whether or not a given packet should be dropped or in some way otherwise affected.

A spin-wait on the system clock's value is done to achieve the best available precision. On the development hardware (SPARC Ultra 10) we achieved about 0.5 ms. Care was taken to ensure the work done within the loop is not computationally expensive (time complexity $\mathcal{O}(1)$, and that work doesn't vary significantly over the course of a run.

ONE's operation reveals characteristics found in both bridges and routers. ONE's operation resembles a bridge as it looks for a match from an IP

packet's destination field in each ARP cache to determine if that packet needs to be forwarded. ONE does not care about the networks each interface resides on as long a the it finds this match.

ONE appears as a router in that it will not implicitly forward packets between its interfaces, when no matching ARP entry is found for the IP packet's destination address.

Deploying ONE without the routers requires some form of priming ONE's own ARP tables. This might simply be initiating an ICMP ping from equipment on each side to some address on the other side.

Because of ONE's operational characteristics, we deployed ONE in the In Space Internet Technology's (ISINT) test bed between a pair of routers as shown in figure V. This configuration allows researchers to add equipment ad hoc, without worrying about priming ONE's ARP caches, or adding additional static ARP entries. Additionally, researchers can more readily utilize ONE in network architectures with asymmetric links, more closely simulating some current methodologies for bulk data transport.

III. Requirements

The platform requirements are: A SPARC running a recent version of Solaris. Development is being done under 5.7. ONE performs best if the host it is running on is fairly fast. Sufficient memory must be present. Available physical memory will constrain queue sizes - causing the host to swap during a run will invalidate any data gathered during that run. A general guideline: to emulate 10BaseT order of magnitude speeds, the platform should be an Ultra-2 or faster.

The host must have at least two network interfaces, and preferably should have two dedicated interfaces. The network segments attached to each of the interfaces should be private and entirely allocatable to the emulation - otherwise unaccounted for traffic can affect the run

The host should be able to be dedicated to running ONE to the exclusion of all else during a run. ONE is CPU intensive. More importantly, ONE needs to have the best possible chance of transmitting packets as close to the time it calculates they must be transmitted as possible in order that the emulation be faithful.

Equipment attached to each of the two interfaces needs to be set up to generate traffic according to research needs. For example: the ISINT test bed where ONE is developed includes a pool of machines on a pair of segments attached to the two interfaces. Applications are run to generate traffic and monitor network performance on the segments as ONE emulates a link between the segments.

To assist in correlation of the data collected from ONE and the traffic generation and measurement tools, we found it useful to precisely synchronize the time on all the involved hosts prior to a run. Some of the measurement tools require close time synchronization. Network Time Protocol (NTP) [3] is an expedient way to accomplish synchronization. Available NTP software can accomplish useful synchronization two ways: either by "jumping" the time as the result of a single synchronization request, or by constantly adjusting the local clock. Either method has side effects that can interfere with getting good test results.

Managing other resources on the ONE host is important. The host should be dedicated to running ONE, to eliminate invalid latency that would otherwise be imposed on the traffic passing through ONE. Steps must be taken to ensure other running processes won't interfere. Non essential processes must be terminated. The critical resources are processor cycles, physical memory, and network bandwidth.

The current version of ONE doesn't take special steps to ensure all required physical memory is present and "allocated" to ONE (i.e. locking the pages). This is because the application places itself on the real time scheduler queue: it's unlikely it will be displaced by any competing process. It is possible that a particular run configuration could cause ONE to request more memory than is physically available, in which case swapping will occur, and subsequently, incorrect timings. An example configuration that would cause oversubscription is: an emulation including a propagation delay of several minutes at 10Mb/s on a host with only 100 MB of physical memory. The page locking capability is anticipated in a future release.

Other considerations apply when deciding how to maintain good time synchronization. Network based NTP is used. Depending on requirements specific to a particular test, accuracy requirements vary. Using ntpdate to "jump" system clocks on all the involved hosts achieved accuracy below 1 msec. (worst case difference between the hosts), but didn't do anything about drift. The commodity PCs that were used for some of the generation and monitoring tasks had fairly inaccurate clocks - it wasn't unusual to see a drift of more than a second over a period of a couple hours. Part way

through the development process, additional hardware was acquired to provide auxiliary "control" network interfaces. NTP peering software (ntpd) was installed and enabled on all the hosts. As noted in the ntpd documentation, a period of time is necessary for the daemons to determine the drifts of all the equipment. One caveat with using the continuously running daemon: attention must be given to the resources it consumes. It communicates over the network, so it should be configured to not utilize any of the test interfaces. The NTP daemons consumes minimal amounts of compute cycles and memory, use of both these resources are insignificant on typical computer equipment.

Another timing issue is: how the Solaris scheduler can potentially affect the accuracy of timing the transmission of packets from either of the output queues. The ONE application uses a combination of sleeps [1] and spin-waits to achieve the best possible accuracy. The spin-waits are necessary as otherwise timing would only be as precise as the system clock rate (rate at which the operating system invokes the scheduler, 10 milliseconds. by default).

Increasing the default Solaris clock tick rate may improve timing accuracy and reduce jitter. The default rate, as mentioned, is 100 Hz. This gives a granularity of 10 ms. The Solaris clock tick rate can be increased to 1000 Hz by adding the line:

```
set hires_tick = 1
```

to the `/etc/system` file and rebooting. A caveat: we found the system to be less stable with the increased clock rate.

It's probably worth mentioning here that the 100 Hz system clock has no relation to the time obtained when a system call to get the time (gettimeofday) is invoked. That time comes directly from a hardware clock that (on an SPARC Ultra 10) has a granularity of 1 μs. (or less - the smallest amount that can be returned by gettimeofday() is 1 μs). Your mileage may vary under other architectures, and this will have to be a consideration when adapting ONE to other architectures.

Variable propagation delay configuration is in part done via a *perl* script. Perl must be present on the host system if variable propagation delay is to be used.

[1] *sleep* in the Solaris/Unix operating system sense - causing the operating system to place the ONE process on the sleep queue and thereby allow other processes to run. If sleeps aren't done occasionally, the ONE host would wedge, as ONE is running on the real time queue and no "ordinary" processes (e.g. all the rest of the processes) would have any chance to run]

The ONE tar file can be unpacked anywhere convenient to the user. The directory it is unpacked to should be added to *$PATH*. The executable (named *one* must be run as effective UID root, as it must access normally protected system resources (specifically, it must be able to access network interfaces promiscuously, and it needs to place itself on the real time run queue). In order to access these resources one of two events has to happen: the user running the application must be logged in as root, or the program itself must be installed setuid root. The usual security caveats apply.

If variable propagation delay is to be used, the one executable must be able to locate the *one-read-raw.pl* script. Place the script somewhere in a *$PATH* directory.

In order that ONE provide a faithful emulation of a network, some requirements must be satisfied. Some of these are self evident. The host running ONE must be fast enough to handle the desired traffic. We found a SPARC Ultra 10 is sufficient for speeds up to around 32 Mb/s. The host should also have sufficient memory to be capable of buffering the worst case (longest) desired latency (at the highest rate).

IV. Configuring and running ONE

A. Invocation

Prior to running ONE, a suitable configuration of the to-be-emulated network must be prepared. The configuration definition is placed in a file, the name of which is specified when ONE is invoked.

ONE is invoked as:

`one [-dn]` *config-file*

where `-dn` is an optionally supplied switch specifying that debugging information be displayed during a run. This option is synonymous with the *verbose* option that can be specified in the configuration file (as described below). The *config-file* parameter, which must be supplied, is the name of the file describing the configuration to be used for this run

B. Configuration Syntax

Items in the configuration file are each specified on a separate line. The format is `keyword: value` where *keyword* specifies a configuration item. Comments may be placed in configuration files by starting comment lines with `#`. The comprehensive list of configuration items is:

- `interface:` *name*

Specify the name of one of the pair of interfaces. *Two* interfaces must be specified, no more, no less.

The interface names must match the names of existing network interfaces on the emulation host (e.g. *hme1*).

- `linespeed:` *value traffic-units* | `infinite`

Specify an upper limit on the rate at which data can flow out a given interface. There must be a pair of these, one following each *interface* specification. The parameter can be either an integer number, representing the maximum speed in *traffic-units* per second, or the identifier `infinite`, indicating no upper limit exists for this interface. The identifier *traffic-units* is described below.

Specifying this limit can affect the flow of data through that interface, causing it to queue as the interface becomes congested. The inherent capabilities of the interfaces still apply - emulating 100BaseT speeds via 10BaseT interfaces isn't likely to yield useful results!

- `memunit:` *value traffic-units*

The internal buffer size (memory allocation granularity) used to store packets in the queue.

- `qsize:` *value traffic-units*

Specify the size of the output queue associated with an interface. There must be a pair of these, one for each interface. The parameters *value*, an integer, and *traffic-units* specify the queue size to ONE. ONE allocates an appropriate amount of memory. No checks are done to ensure sufficient physical memory is available, so some attention to this detail is necessary.

- `red-threshold:` min_{th} max_{th} *units-designator*

Enable *random early detection* (RED). RED [2] is an algorithm used to ensure fair access to network bandwidth in situations where congestion is causing packets to be dropped. If not enabled, ONE will discard new traffic if the queue the traffic is bound for is full.

When RED is enabled, a pair of parameters must be supplied. The pair of numbers ranging from 0.0 to 1.0 represent *queue fullness* and can be thought of as the points at which a zero and a 100 percent chance that a random packet will be dropped from the outbound queue for each incoming packet.

For example: `red-threshold 0.75 0.95 b` is interpreted as: when this output queue is less than 75 percent full do not discard any packets. When the queue increases above 75 percent, using a linear probability (zero percent chance at 75 percent queue full to 100 percent chance at 95 percent queue full) to determine if a packet should be dropped. If a packet is to be dropped, pick one randomly from the output queue.

The *units-designator* is *p* or *b* to indicate packets or bytes respectively.

- `ber:` *value*

Specify a *bit error rate* (BER). If specified, pseudo-noise will be injected into the traffic stream. The configuration item allows for a simple specification of noise which will cause random single bits to be obscured at the specified probability. The parameter is a number representing the probability (0.0 to 1.0) that a given bit will be obscured by noise. This item is specified per interface.

- `propagation:` *value* `[ variable`*file* `]`

Specify the propagation delay that is to be applied to a stream of traffic before it can be transmitted through the output interface. In the implementation, traffic is queued on the output queue, but is not transmitted until the specified delay interval has passed. Caveats about appropriate queue size vs. having traffic discarded apply. The parameter *value* is a floating point number and represents delay time in milliseconds. If the optional parameter *variable* is specified, the specified file is read to obtain the variable propagation delay values. The variable propagation delay values are added the fixed value. Variable propagation delay is useful in instances like configurations of satellites in a network, where propagation delay is not fixed. If variable propagation needs to be specified, the input format is described below.

- `verbose:` *level*

Enable verbose status reporting. Additional information is displayed and logged if *verbose* is enabled. The parameter *level* can range from one to ten, with higher levels resulting in increasing amounts of logged information. A caveat: logging large amounts of information may adversely affect the veracity of a run, as the act of logging consumes resources. Actions are:

0 - dump counters
1 - dump cache, interface information, Ethernet cache
2 - dump print queue
3 - dump qfull
4 - dump printdrops
5 - dump qfull2
8 - dump delays

- `drawplots`
- `logfile:` *name*

Specify the name of the file to which the logs will be written. The file is overwritten each run, so if the information within the file is to be saved, it will have to be copied elsewhere.

- `lockfile:` *name*

Specify the name of the lock file - a file used to help

ensure ONE cannot be run more than once simultaneously on a given host. On occasion, if ONE isn't terminated gracefully, the lock file might have to be manually removed.

- `pidfile:` *name*

ONE writes its PID in this file while running.

- `delayfile:` *name*

Specify the name of the file to which the delay results are to be written. Note that the verbose level must also be set to eight or greater for this to happen.

- `drop`

Currently, this item has no effect.

The parameter *traffic-units* is an identifier comprised of one of the following optional characters:

`K` - 1024 (2^{10})
`k` - 1000
`M` - 1048576 (2^{20})
`m` - 1000000

followed by one of:

`B` - bytes/octets
`b` - bits

V. Using ONE

ONE is set up on a computer with a couple of interfaces (ideally three interfaces, the third to provide a route for traffic not intended for the network emulation.

If the varying propagation capability is to be used, it's best to synchronize ONE's operation with that of the traffic generators (and the measurement instruments). It's fairly straightforward to do this with a script. ONE, when operating in this mode, will wait until it first sees traffic before starting the propagation model (in other words, time t_0 is the time of the first traffic seen).

An example configuration file:

```
# Satellite Emulation

# An example configuration for a
# host that has two dedicated
# interfaces, le0 and le1. The
# verbosity of diagnostics is set to a
# relatively quiet level (one);
# only counters, cache, interface
# information, and the Ethernet
# cache will be dumped.

#
# propagation: ms
#    the propagation delay in milliseconds
# qsize unit: size of the outgoing packet queue
#   can use units below, no units=bytes
# linespeed unit: variable
#   can use units below, no units=bytes/second
#   a linespeed of "infinite" means go as
#    fast as possible
# memunit:  memory allocation size for buffering
#    packets, same units as linespeed (1K means
#    alloc packets in 1K allocation units)
# verbose:  number for 0 to 10, 0 is quiet, 10 is noisy
# --------------------
# values that take args in bits/bytes can use the
# following units: K=1024, k=1000, b=bits, B=bytes,
# M=1024*1024, m=1000*1000, s=ignored so
# 1 Kbs = 1024 bits/second, and 1 kBs=1000 bytes/second

verbose: 1
logfile: /tmp/bridge.log
pidfile: /tmp/bridge.pid
delayfile: /tmp/bridge.delay
memunit:     1024 B
drawplots:  0

# front side

# The maximum rate at which data
# can exit this interface is set to
# 768000 bits per second or 96000
# bytes per second. The queue size is
# limited to 85 KB (87040 bytes).
# Propagation delay is set to 50
# milliseconds; traffic entering this
# interface will be delayed at
# least 50 milliseconds before
# exiting the other interface. Random
# Early Detection is enabled and is
# set to start discarding random
# packets when the transmit queue is
# 70 percent full and to discard on
# a one for one basis if the queue
# reaches 95 percent full.

interface: le0
  linespeed:   768000 bs
  qsize:       85 KB
  propagation: 50
  drop: no\_drop
  red-threshold: 70 95 p

# back side

# Configured pretty much symmetrically,
# with the exception RED is not
# enabled. If the queue becomes full,
# new packets will be dropped.

interface: le1
  linespeed:   768000 bs
  qsize:       85 KB
  propagation: 50
  drop: no\_drop
```

A variety of example configurations can be found in the *config* directory.

The variable propagation delay input file format is, for expediency, that of the output from the Satellite Tool Kit application (STK). The format is a series of records, one per line. Fields within a given record are separated by whitespace.

An example from a variable propagation delay file:

```
1 Mar 1999 00:26:14.00 228.443  -81.139  44708.216092
1 Mar 1999 00:27:14.00 226.128  -80.972  44413.993235
1 Mar 1999 00:28:14.00 223.845  -80.819  44113.620746
1 Mar 1999 00:29:14.00 221.587  -80.682  43808.004479
...
```

The first four fields represent the date, and the last (seventh) field represents the distance between the satellite and the observer in kilometers. The other two fields are not used by ONE.

Runs using variable propagation delay are based on the start time of the run being treated as the start time of the delay data in the propagation file. Thus, for example, using the above data, the delays imposed on traffic by ONE one minute into a run would be based on the second record, and so on. When ONE reaches the end of the variable delay propagation table the run is terminated. No interpolation is done of delays between data points supplied in the table.

The propagation delay information is stored as a table internally. Each entry consumes around 12 bytes. Choice of an appropriate interval between time data points is a trade off between maximum acceptable jumps in the delay value, and the size of the table (specifically, how the size impacts the amount of available memory). Table size has no significant effect on compute resources

Zones of exclusion (ZOEs) are determined based on large discontinuities in the *time* column of the delay table, that is, when there is a step in the time of greater than one minute. During the time covered by a ZOE occurrence, no traffic is forwarded.

Example: measuring jitter between two hosts through ONE.

In this example, ICMP pings are used at a relatively low bandwidth utilization (so link saturation is not an issue) as a simple test to verify the test traffic throughput is as expected, and not being adversely affected by some part of the test apparatus. The hosts participating in the test are terrestrial2 and space2. Host terrestrial2 pings host space2 and measures the response times.

Th script:

```
#! /usr/local/bin/perl

use strict;

my $interval = 0.2;
my $size = 64;
my $count = 100;
my $shost = "terrestrial2";
my $dhost = "space2";

my $rtmp = "/tmp/ping.$$";
close STDIN;
my $ocmd = "ssh root\@$shost-ext \
 'ping -i $interval -s $size -c $count $dhost > $rtmp; \
 cat $rtmp'";
print STDERR "$ocmd\n";
open(STDIN, "$ocmd |") || do {
  die "open, $!";
};
my $n_rtt = "/tmp/plot-$$-rtt.ps";
open(RTT,
  "| graph -T ps \
 -L 'RTT, $shost to $dhost, $size B, $interval S' \
 -X 'Count (time)' -Y 'Propagation Delay (msec)' \
 > $n_rtt");
my $n_drtt = "/tmp/plot-$$-drtt.ps";
open(DRTT,
  "| graph -T ps \
 -L 'd(RTT), $shost to $dhost, $size B, $interval S' \
 -X 'Count (time)' -Y '(msec)' > $n_drtt");
my $prev_seq = -1;
my $ortime;
while (<STDIN>) {
  chomp;
  if (/icmp_seq=(d+) .+ time=([d.]+) /) {
    my $icmp_seq = $1;
    my $rtime = $2;
    while ($icmp_seq != $prev_seq + 1) {
      my $mtime = $interval * $prev_seq;
      print "missing at time $mtime\n";
      $prev_seq++;
    }
    my $time = $interval * $icmp_seq;
    print RTT "$time $rtime\n";
    if ($icmp_seq > 1) {
      printf(DRTT "%g %g\n", $time, $rtime - $ortime);
    }
    $prev_seq = $icmp_seq;
    $ortime = $rtime;
  }
  else {
    print STDERR "warning: ignoring '$_'\n";
  }
}
close RTT;
close DRTT;
if ($prev_seq == -1) {
  die "no data";
}
system("gv $n_rtt &");
system("gv $n_drtt &");
```

VI. Future Plans

Other algorithms for queuing strategies will be incorporated into ONE. Something that attempts to restrict use of a link by non rate limiting traffic (UDP, for instance) would be useful.

The ONE application was developed in a Solaris/SPARC environment. Near term plans include porting it to a Solaris/X86 environment (which shouldn't be much more than a recompile). Software development tools include the ubiquitous GNU C compiler. There is some reliance on low level Solaris facilities that will make it a bit more challenging to make a platform independent version, but that is a longer term goal.

Running on less expensive commodity hardware does have some caveats. Depending on the quality of both the hardware and the software (device drivers and such), performance can vary widely. Inexpensive PCI based Ethernet interfaces can have poor to excellent performance, depending on the production run (i.e. even identical model numbers from the same manufacturer). When running on "commodity" hardware, it is important to quantify the performance of the underlying hardware and software before relying on ONE to produce correct emulation.

VII. Availability

ONE lives at `http://irg.cs.ohiou.edu/one/`. Source code, documentation and other information related to ONE can be found at this address.

A mail list has been established to facilitate communication among ONE users. Send email to `majordomo@grc.nasa.gov`. In the text body include the line:

```
subscribe one-users
```

to subscribe to the list.

References

[1] Mark Allman, Adam Caldwell, Shawn Osterman *ONE: The Ohio Network Emulator* Ohio University, 1997.

[2] Sally Floyd, Van Jacobson *Random Early Detection Gateways for Congestion Avoidance* IEEE/ACM Transactions on Networking, 1993.

[3] ntpd, ntpdate http://www.eecis.udel.edu/ñtp/